AutoRGNN: um Modelo de Previsão de *Churn* que Preserva a Privacidade com uma Abordagem Híbrida de Aprendizado Profundo

Gabriel Teixeira Pinto Coimbra
Universidade Federal de Viçosa
Viçosa, Brasil
gabriel.coimbra@ufv.br

Mateus Pinto da Silva
Universidade Federal de Viçosa
Viçosa, Brasil
mateus.p.silva@ufv.br

Fabrício Aguiar Silva
Universidade Federal de Viçosa
Viçosa, Brasil
fabricio.asilva@ufv.br

Thais Regina de Moura Braga
Universidade Federal de Viçosa
Viçosa, Brasil
thais.braga@ufv.br

RESUMO

In recent years, the use of mobile applications for digital service is being widely deployed in a varied range of contexts. With this, predicting the possibility of churn is vital for selecting users that could be targeted with user-retention campaigns. This technique is commonly referred as Churn Prediction Problem (CPP). Most studies in the literature use traditional machine learning techniques to predict churn, and neglect the users' privacy. In this work, we propose a privacy-preserving solution that uses neural network to predict churn of mobile services. Our solution, called AutoRGNN, requires only the installation and uninstallation sequences of mobile apps, and integrates Recurrent and Graph Neural Networks. In comparison with a traditional baseline approach in a large-scale and real scenario, AutoRGNN was capable to increase the recall and precision up to 19% and 7%, respectively.

KEYWORDS

CHURN PREDICTION, PRIVACY PRESERVING, AUTOENCODER, RECURRENT NEURAL NETWORKS, GRAPH NEURAL NETWORKS

1 INTRODUÇÃO

Empresas prestadoras de serviços por meio de aplicativos móveis são afetadas com o abandono de clientes [5], que é chamado de *churn* na literatura [3]. Além de ações para conseguir cada vez mais novos clientes, é importante evitar que os clientes atuais desistam do serviço. Uma estratégia comum é a utilização de programas de fidelização oferecidos a todos os clientes. Porém, esses programas podem ter um custo financeiro alto, pois as empresas utilizam recursos com clientes satisfeitos e que não desejam realizar *churn*. Assim, uma alternativa que tem ganhado destaque é a de antecipar quais são os clientes mais propensos a desistirem do serviço, ou seja, predizer quais clientes provavelmente realizarão o *churn* [3]. Dessa forma, campanhas de fidelização podem focar nesses clientes, e evitar despesas desnecessárias com clientes que dificilmente fariam *churn*. Além disso, é possível tentar prever o *churn* que irá acontecer em curto, médio ou longo prazo, sendo que a melhor escolha depende das necessidades da empresa.

O problema de previsão de *churn* já vem sendo explorado na literatura em serviços de telefonia/Internet e cartões de crédito, que apresentam altas taxas de *churn*, principalmente utilizando técnicas tradicionais de aprendizado de máquina supervisionado. Em geral, nestas soluções dados sensíveis dos clientes são utilizados, como localização, movimentações financeiras, dados pessoais, uso detalhado dos serviços, dentre outros. Contudo, muitos países, inclusive o Brasil (com a Lei Geral de Proteção de Dados - LGPD), avançam suas legislações e dificultam o uso de dados sensíveis para protegerem a privacidade dos usuários. Por isso, a coleta e o armazenamento desses dados se torna mais complicado [4]. Entretanto, não usar dados que sejam sensíveis aos clientes pode reduzir significativamente a qualidade da previsão dos clientes que farão *churn*.

Dito isso, a hipótese de pesquisa deste trabalho é que o uso de técnicas de aprendizado profundo pode ser uma alternativa interessante, pois essas técnicas tendem a oferecer uma melhor acurácia, o que pode compensar a falta de dados sensíveis. O objetivo deste trabalho é desenvolver uma solução de aprendizado profundo com modelos híbridos que preserve a privacidade dos usuários para a previsão de *churn* de serviços digitais. A solução proposta, chamada AutoRGNN, é um modelo híbrido composto de redes neurais recorrentes (RNNs) autocodificadoras, redes neurais de grafos (GNNs) e redes neurais tradicionais (i.e., *feed-foward*). Serão utilizados apenas os dados de instalação e desinstalação de aplicativos móveis, sem a necessidade de nenhum dado sensível como localização dos clientes, ou ainda informações pessoais ou financeiras, tornando a solução generalista para qualquer aplicativo móvel.

Como estudo de caso, foram utilizados dados reais de 47.665 clientes de um banco digital do Brasil. A solução proposta em aprendizado profundo foi comparada com uma abordagem tradicional utilizando um modelo obtido com uso de AutoML pela ferramenta TPOT [12]. Os resultados revelaram que foi possível melhorar a revocação e a precisão da classificação de *churn* em até 7% e 19%, respectivamente.

As principais contribuições deste trabalho são: a) utilização de dados não invasivos, mas generalistas, para previsão de *churn*, algo pouco explorado na literatura. b) utilização de técnicas recentes de aprendizado profundo (i.e., GNN) para codificação de sequências de eventos em conjunto com métodos tradicionais (i.e., Autoencoder RNN). Nossos resultados são promissores, indicando que a sequência de instalação e desinstalação de aplicativos é uma fonte importante de dados não invasivos para detecção de *churn*.

O restante deste texto está organizado da seguinte forma: a Seção 2 descreve os trabalhos relacionados. A Seção 3 apresenta a proposta deste trabalho. A Seção 4 trata dos dados, avaliações e resultados obtidos. Por fim, a Seção 5 discorre sobre as considerações finais.

2 TRABALHOS RELACIONADOS

O contexto mais comum para utilização de soluções automáticas de previsão de usuários *churners* é quando a taxa de *churn* é alta. Isso é percebido no setor de telecomunicações (telefonia, Internet e televisão) [10, 19], mas outros contextos ainda incluem: aplicativos de redes sociais [22], aplicativos de *streaming* de música [23], de cursos online [17] e de empresas fornecedoras de cartão de crédito [15]. Além desses contextos, também é possível encontrar trabalhos que investigam a previsão de *churn* de bancos [6, 14].

Em todos os trabalhos encontrados foram utilizados dados invasivos que necessitariam de medidas para proteção ao armazenar e transmitir. Além disso, é importante ressaltar que essas medidas variam dependendo da legislação do país ou estado, como por exemplo a GDPR na Europa e LGPD no Brasil. Além disso, essas soluções tendem a não ser generalizáveis a outros contextos dos que os apresentados, pois utilizam dados específicos das plataformas em que foram testadas. Por exemplo, no contexto bancário, o mais comum é uso do histórico transações, *score* de crédito, saldo e informações sociodemográficas. Não seria possível utilizar esse tipo de informação para prever *churners* de aplicativos de *streaming* de música [2].

O uso de informações sequenciais em algoritmos com capacidade de utilizá-las sem pré-processamento é feito em [17, 22, 23]. Nos demais, as informações sequenciais são utilizadas após algum pré-processamento que as tornem informações estáticas para que sejam utilizadas em algoritmos tradicionais de aprendizado de máquina [6, 10, 14, 19].

A Tabela 1 apresenta um resumo dos principais trabalhos encontrados. Além destes, em [21] são apresentadas formas de preservar informações dos usuários com distorções aplicadas nos dados antes da previsão, mas para isso é necessário ter acesso aos dados confidenciais, o que não é preciso na solução proposta por este trabalho. Ademais, outra forma com contexto similar a este trabalho [7] é utilizar parâmetros dos modelos treinados com dados confidenciais do passado ao invés de utilizar os dados diretamente, mas, ainda assim, são necessários os dados confidenciais. Além destes trabalhos, não foram encontrados trabalhos de previsão de *churn* de serviços digitais em forma de aplicativos móveis que sejam generalistas e dispensem informações confidenciais.

Diferentemente dos trabalhos existentes, neste trabalho as técnicas de aprendizado profundo são utilizadas para prever *churn* de serviços móveis sem utilizar dados sensíveis dos clientes. Dos usuários, apenas as informações de instalação e desinstalação de aplicativos é utilizada.

3 AUTORGNN

Neste trabalho, cada cliente é representado como uma sequência de eventos, onde cada evento representa a instalação ou desinstalação de algum aplicativo. O evento contém um identificador numérico único para cada aplicativo, a data de ocorrência e um valor binário que indica se é uma instalação ou desinstalação. Com isso, o conjunto de dados é definido da seguinte forma:

Definição 1 (Conjunto de dados). *Seja E_u as sequências de eventos ordenadas cronologicamente para o cliente u. Cada $e_i \in E_u$ é uma tupla $\langle \alpha, \beta, \theta \rangle$ onde α especifica um número único de identificação para cada aplicativo móvel, β é um inteiro especificando o dia que este evento aconteceu, começando de um a partir do início do registro dos dados para o usuário u, e θ é um indicador binário que representa a instalação ou desinstalação de um aplicativo. É importante destacar que, para todos os usuários no dia $\beta = 1$, existem somente instalações, pois são os aplicativos instalados no smartphone do usuário antes do começo do registro de instalações e desinstalações.*

A seguir, a proposta do trabalho chamada AutoRGNN é apresentada. Ela é composta por uma Rede Neural Recorrente Autocodificadora (AutoRNN) apresentada na seção 3.1 e uma Rede Neural de Grafos (GNN) apresentada na seção 3.2. A integração das duas no modelo híbrido é descrita na seção 3.3.

3.1 Redes Neurais Recorrentes Autocodificadoras (AutoRNN)

Os dados sequenciais da definição 1, ao serem adaptados para algoritmos de aprendizado de máquina, tendem a ter uma estrutura esparsa, devido ao grande número de aplicativos possíveis de serem instalados e a baixa quantidade de eventos por dia. Portanto, empiricamente mostrou-se mais interessante a utilização de um modelo híbrido para representar a sequência de eventos de uma forma alternativa que eliminasse a esparsidade. Soluções como o PCA (*Principal Component Analysis*) são comumente utilizadas nesses contextos, porém o PCA não é capaz de lidar diretamente com dados sequenciais. Portanto, neste trabalho foi utilizado uma rede neural recorrente autocodificadora para uma representação não-esparsa dos dados sequenciais. Para a solução proposta nesse trabalho, essa representação não-esparsa foi utilizada com a saída do modelo utilizando redes neurais de grafos, descrita na seção 3.2 e a combinação desses algoritmos com a rede neural *feed-forward* utilizando dados estáticos na seção 3.3.

Diferentemente das redes neurais *feed-foward* (FNN) (i.e., densamente conectadas), as redes neurais recorrentes usam um formato de entrada que codifica a sequência de passos temporais. As FNNs trabalham com um tensor $t \in \mathbb{R}^{(\gamma,\tau)}$ em que γ representa o número de amostras de treinamento e τ representa os dados associados a cada amostra, que pode ter codificação binária ou mista. Para RNN, é necessária uma dimensão extra, e a forma do tensor t é a seguinte: $t \in \mathbb{R}^{(\gamma,\delta,\tau)}$, em que δ representa o passo temporal. Portanto, para codificar a base de dados da definição 1, inspirou-se do uso bem-sucedido de RNNs da literatura em problemas de processamento de linguagens naturais. Para isso, podemos presumir que existem padrões que podem ser capturados pela sequência de instalações e desinstalações assim como em textos em linguagem natural que permitam diferenciar os usuários *churners* dos usuários não *churners*. A seguinte representação é proposta para permitir o aprendizado desses padrões:

Definição 2 (Dados de entrada da Rede Recorrente Autocodificadora (X_{rnn})). *Como pode ser visto na figura 1a, foi definido o*

Trabalho	Nº de clientes	Proporção de *churn*	Atributos	Métodos	Contexto
[6]	50000	22%	Tipo de conta, transações, saldo, renda, sociodemográfico	BiLSTM	Bancário
[14]	46406	10%	Tipo de conta, transações, saldo, renda, sociodemográfico	SVM,KNN, Árvore de Decisão, Floresta aleatória	Bancário
[10]	3,333	14%	Comportamento de uso da rede de telefonia pelo usuário	*Boosting + Logistic Regression*	Telecom
[19]	5 milhões	*	Número de chamadas, mesnagens, uso da rede telefonica, tipo de plano, sociodemografico	*EasyEnsemble*	Telecom
[22]	40 milhões	Desconhecido	Comportamento de uso diário da plataforma	LSTM, *Kmeans*, *Attention*	Music streaming
[23]	156 mil	13%	Comportamento de uso diário da plataforma, sociodemográfico	LSTM + CNN	*Streaming* de música
[17]	120 mil	80%	Comportamento de uso diário da plataforma, sociodemográfico	*Blended Learning Approach* (LSTM)	Cursos online
Este trabalho	47 mil	27%	Instalação e desinstalação de applicativos	Autoencoder + BiGRU + GNN	Bancos digitais

Tabela 1: Resumo dos trabalhos relacionados.

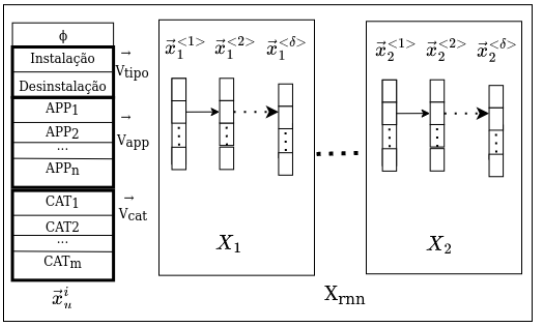
conjunto de treinamento da RNN Autocodificadora como sendo o tensor $X_{rnn} \in \mathbb{R}^{\gamma,\delta,\tau}$. *O tensor* X_{rnn} *pode ser decomposto por uma matriz* X_u *para cada usuário, onde* $X_u = (\vec{x}_u^{<1>}, \vec{x}_u^{<2>}, \ldots, \vec{x}_u^{<\delta>})$. *Cada passo temporal representa um evento codificado como um vetor* $\vec{x}_u^{<i>}$, *definido pela concatenação de vetores* $\vec{x}_u^{<i>} = ((\phi)\|\vec{V}_{tipo}\|\vec{V}_{app}\|\vec{V}_{categoria})$, *onde* ϕ *indica em qual momento o evento aconteceu, relativo aos dias de registro do usuário. Já os vetores* $\vec{V}_{tipo}, \vec{V}_{app}, \vec{V}_{categoria}$ *denotam vetores com codificação binária (i.e., one-hot) para o tipo de dados* $p = \{tipo, app, categoria\}$. *Além disso, para cada vetor presente em* $\vec{x}_u^{<i>}$ *é efetuada a restrição* $\|\vec{V}_p\|_2 = 1$.

Na definição 2, o valor ϕ é utilizado para codificar a passagem de tempo. Por exemplo, se um usuário u tem D_u dias de dados registrado e no dia β_i instalou o aplicativo de número α_j, o valor de ϕ que codifica esse evento será β_i/D_u. A divisão por D_u é necessária para normalizar os valores para facilitar o aprendizado da rede. Já a concatenação de vetores presente na definição 2 é utilizada para codificar os eventos da seguinte forma: o vetor $\vec{V}_{tipo}$ codifica o tipo de evento (i.e., instalação ou desinstalação) e tem valor $\vec{V}_{tipo} = (1, 0)$ caso represente uma instalação e, analogamente, $\vec{V}_{tipo} = (0, 1)$ caso seja represente uma desinstalação. Esse tipo de representação é comum para dados categóricos e também é utilizada para representar qual aplicativo será instalado usando o vetor $\vec{V}_{app}$ e a categoria deste aplicativo com o vetor $\vec{V_{cat}}$. Nesses últimos dois vetores, somente uma posição tem valor 1, pois somente um aplicativo pode ser instalado por evento sendo permitida somente uma categoria por aplicativo.

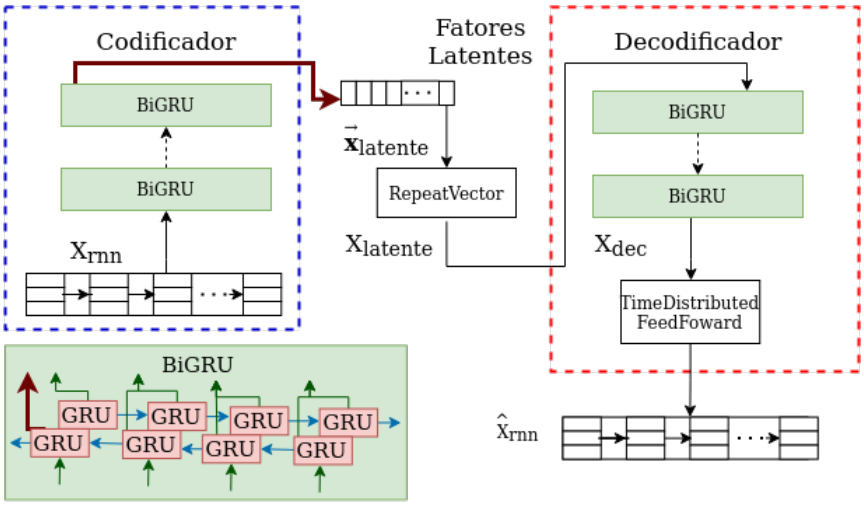
A RNN Autocodificadora (AutoRNN), ilustrada na figura 1b, tem como objetivo de minimização: $\arg\max_{\hat{X}_{rnn}}(\hat{X}_{rnn} - X_{rnn})$ onde $\hat{X}_{rnn} = f(g(X_{rnn}))$, f e g são funções abstraindo os módulos de codificação e decodificação, respectivamente, e $\hat{X}_{rnn}$ é a aproximação obtida de X_{rnn}. Para o treinamento desse autocodificador, o conjunto de treinamento X_{rnn} é passado como entrada para codificador abstraído pela função f que produz o vetor $\vec{x}_{latente}$. Este vetor é futuramente utilizado para a previsão de *churn*, mas na AutoRNN ele é adaptado para uma matriz pela função *RepeatVector* para ser passado como entrada para o decodificador abstraído pela função g. O treinamento deste modelo produz o vetor $\vec{x}^u_{latente}$ que é uma representação densa da sequência X_{rnn} para o usuário u, eliminando a esparsidade encontrada em X_{rnn}. O vetor $\vec{x}_{latente}$ é posteriormente utilizado para o aprendizado supervisionado definido na seção 3.3. A codificação e decodificação é feita utilizando camadas de RNNs bidirecionais [16] empilhadas. Além disso, as células recorrentes são da variante *Gated Recurrent Unit* (GRU) [1]. Devido à falta de espaço, para detalhes sobre os cálculos envolvidos nessas camadas o leitor é convidado a ler os trabalhos citados. Além das camadas GRU, também são utilizadas as camadas *RepeatVector* [1] e *TimeDistributed* [2] para adaptar as dimensões dos dados. A primeira pode ser descrita por uma concatenação de vetores $RepeatVector(\vec{v}, k) = (\vec{v}^t; \vec{v}^t; ...; \vec{v}^t)$, onde $RepeatVector(\vec{t}, k) \in \mathbb{R}^{k,\zeta}$ e o vetor $\vec{v} \in \mathbb{R}^{\zeta}$. Esse vetor é obtido dos pesos da saída da última célula na propagação inversa de uma RNN bidirecional [16]. Ressaltamos que $|\vec{x}_{latente}| = \zeta$ é a quantidade de unidades na célula recorrente conectada à função *RepeatVector*. Essa função é utilizada para adaptar o vetor $\vec{x}_{latente}$ para a matriz de entrada do decodificador $X_{latente}$. Já a segunda função, $TimeDistributedFeedFoward \in \mathbb{R}^{k,\zeta}$ permite o aprendizado de δ camadas *feed-foward*, aplicada independentemente em cada passo temporal e permitindo a reconstrução $\widehat{X}_{rnn}$ da matriz X_{rnn}.

Definida a AutoRNN, é necessário utilizar alguns filtros para adaptar o conjunto de dados da definição 1 para o tensor X_{rnn}

[1] https://keras.io/api/layers/reshaping_layers/repeat_vector/
[2] https://keras.io/api/layers/recurrent_layers/time_distributed/



(a) Formato da entrada da AutoRNN para o usuário u.



(b) Arquitetura da AutoRNN. A seta em vermelho indica uso da saída ($\hat{y}$) da última célula bidirecional após o processamento de toda a sequência. As setas pontilhadas indicam a possibilidade de repetição de camadas para aumentar a profundidade do modelo.

Figura 1: Arquitetura da AutoRNN e dados da AutoRNN

evitando dados esparsos. Isso porque a quantidade de aplicativos únicos é proporcional ao número de parâmetros e a quantidade de dados necessários para uma boa acurácia do modelo sem sobre-ajustamento. Portanto, na seção 4 é mostrado um pré-processamento para filtrar usuários com sequências muito longas ou muito curtas na base de dados. Esse passo é necessário para evitar uso excessivo de preenchimento (i.e., *padding*, valores nulos no tensor) necessário para todas matrizes X_u terem a mesma dimensão (δ, τ). Após esse filtro, o valor de δ corresponde ao tamanho da sequência do usuário com o maior número de sequências da base de dados.

3.2 Redes Neurais de Grafos (GNN)

A codificação descrita na definição 2, apesar de intuitiva e bem descritiva do conjunto de dados, pode não promover o aprendizado de padrões adequado para a tarefa de previsão de *churn*. Portanto, uma codificação alternativa não-esparsa dos dados da definição 1 é apresentada nesta seção e utilizada em uma GNN. A partir da GNN descrita nesta seção é possível obter mais um vetor latente para cada usuário que pode ser utilizado junto com o vetor $x_{latente}$ da seção 3.1.

Para a GNN, é necessário definir um grafo a partir dos dados da definição 1. Nesta abordagem, os grafos são gerados independentemente para cada usuário u. Para isso, é criado um grafo como matriz de adjacência para o usuário u no formato $A^u \in \mathbb{R}^{\psi,\psi}$ onde $\psi = |\vec{V}_{app}|$. A matriz A^u é inicializada com todas as arestas com o peso 0, produzindo um grafo desconexo. Todas as tuplas apresentadas na definição 1 são agrupadas em grupos g_β conforme a data do evento β. Como pode ser visto na figura 2, os valores da matriz A^u são incrementados conectando todas as tuplas de um mesmo grupo, pois assume-se que os aplicativos instalados e desinstalados no mesmo dia têm um relacionamento que pode ser utilizado para diferenciar *churners*. Assim, o peso das arestas correspondentes a esses aplicativos são incrementados utilizando a função: $A_{k,l} := A_{k,l} + P(k) \times P(l)$, onde k e l representam aplicativos de todas as tuplas um grupo g_β, ou seja, são instalações e desinstalações de aplicativos que aconteceram no mesmo dia. A função P define o peso da tupla, que é -1 se for uma desinstalação (ou seja, se o binário θ for 0), e 1 se for uma instalação de aplicativo (se o

binário θ for 1). Dessa forma, é esperado que dois aplicativos que foram desinstalados ou instalados tenham uma correlação positiva, e que um aplicativo instalado e outro desinstalado tenham um peso negativo.

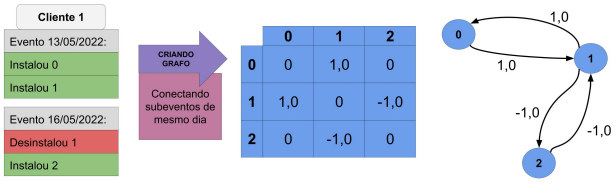


Figura 2: Exemplo de grafo.

Os grupos de tuplas são ordenados crescentemente pelo dia β, e cada grupo de tuplas g_β é considerado em par com seu próximo $g_{\beta+1}$. Então, a matriz de adjacência A^u para o usuário u tem seus pesos normalizados segundo a função $A^u_{m,n} := P(m) \times P(n)/(\beta_m - \beta_n + 1)$. Sendo m todas as tuplas do primeiro grupo considerado (g_β), n todas as tuplas do grupo subsequente em data ($g_{\beta+1}$) e $\beta_m - \beta_n + 1$ representa a quantidade de dias corridos entre β_m e β_n. Assim, as relações entre instalações e desinstalações são enriquecidas com a informação da distância temporal entre as datas (i.e., é atribuído um peso proporcional para eventos próximos cronologicamente). Além da distância temporal, em cada nó também são armazenadas informações do aplicativo como: número de instalações, categoria e *score* de avaliações na *Google Play*.

As GNNs permitem a possibilidade de, além de armazenar pesos na matriz de adjacências, também é possível armazenar um vetor com dados em cada aresta e outro vetor em cada nó. Nesta solução, foram armazenados dados que descrevem as características dos aplicativos reportados pela *Google Play Store* como: número de instalações, categoria e *score* de avaliações. Com isso é feita a matriz $N^u \in \mathbb{R}^{\psi,\lambda_1}$ para o usuário u onde λ é o número de atributos do nó (i.e., aplicativo) e cada valor é normalizado utilizando a normalização *Z-score* (i.e., $media(N_{:,k}) \approx 0$ e $stdev(N_{:,k}) \approx 1$).

Após a definição dos grafos, existem várias possibilidades de arquiteturas das GNNs, pois é possível combinar camadas convolucionais com camadas de *pooling*. Neste trabalho foi utilizada a camada *Graph Attention layer*(GAT) [18]. A GAT é abstraída pela função $GAT(A^u, N^u) = \vec{N}'^u$ onde A^u e N^u são as matrizes de adjacência e atributos de nó para o usuário u, respectivamente, e $\vec{N}'^u \in \mathbb{R}^{\psi,\lambda_2}$ representa o resultado da convolução dos grafos sendo λ_2 o número de canais de convolução escolhido por meio de otimização de hiperparâmetros. A saída da GAT é transformada em vetor para ser utilizada no modelo AutoRGNN utilizando a concatenação de vetores: $\vec{x}^u_{gnn} = [N'^u_1, N'^u_2, ..., N'^u_\psi]$.

3.3 Combinando a AutoRNN e GNN com dados estáticos

Além das codificações latentes obtidas pelos modelos descritos nas seções 3.1 e 3.2, também são utilizados alguns dados para cada usuário que não fazem parte de uma sequência temporal. Em primeiro lugar, é utilizado um vetor com codificação binária (i.e., *one-hot*) com a lista de aplicativos instalados no último dia de coleta de dados para cada usuário. Neste caso, uma quantidade ainda menor de aplicativos é utilizada devido a facilidade com que redes neurais profundas sobre-ajustariam nesses dados esparsos. Portanto, utilizamos o método *Sequential Feature Selection* com auxílio do modelo produzido pelo algoritmo Floresta Aleatória para escolher os aplicativos que, ao serem removidos, não reduzirão a acurácia do modelo significativamente. Também é acrescentado a esses dados os seguintes atributos calculados utilizando a base de dados definida em 1: quantidade de dias únicos disponíveis, quantidade de dias do primeiro registro até o último, quantidade total de aplicativos desinstalados e quantidade total de aplicativos instalados. Esses dados calculados a partir da sequência são representados no vetor x_{static} da figura 3. É importante destacar que as informações contidas no vetor $\vec{x}_{static}$ também estão disponíveis ao modelo da *baseline*.

Com isso, o modelo final para classificação de *churn* é ilustrado na figura 3. Cada vetor latente (i.e., $\vec{x}_{latente}, \vec{x}_{gnn}, \vec{x}_{static}$) codifica o usuário de formas diferentes. Estes são passados para camadas denominadas FNN compostas de redes neurais *feed-foward* densamente conectadas com *BatchNormalization* [9] e *Dropout* [20]. A quantidade de unidades em cada camada, assim como a quantidade de camadas FNN (i.e., profundidade) do modelo denotada pelos valores $p_i \in \mathbb{N}^*$, a taxa de *dropout*, são otimizadas utilizando soluções automáticas. Além disso, é utilizado uma *skip connection* no estilo *DenseNet* [8]. No fim, é utilizada uma camada *sigmoid* com o objetivo de classificar os *churners* e com isso, o modelo é treinado otimizando a função que minimiza a diferença entre *churners* reais e *churners* previstos usando a função de entropia cruzada.

4 AVALIAÇÃO

4.1 Os dados

Para avaliar a proposta, foram utilizados dados reais de usuários de um banco digital no Brasil como estudo de caso, obtidos com uma empresa parceira por meio de um acordo de confidencialidade. A coleta foi feita por um software agente integrado com o aplicativo de um banco digital no *smartphone* do cliente. Portanto, a coleta dos dados usados neste trabalho começa quando o cliente instala o aplicativo e passa a utilizá-lo. O software agente captura a lista de aplicativos instalados no *smartphone*. Por isso, assume-se que estão disponíveis apenas quais aplicativos estão no *smartphone* ao longo do tempo, além de informações das categorias dos aplicativos. Com isso, é importante destacar que não há garantias de que cada *smartphone* irá reportar os aplicativos instalados em todos os dias. Portanto, é possível que haja instalações e desinstalações não presentes na base de dados pois aconteceram antes que o *smartphone* enviasse o relatório contendo os aplicativos instalados. Além disso, também é importante destacar que, se mais de um aplicativo foi instalado ou desinstalado em um mesmo dia, não se sabe a ordem que esses eventos aconteceram. Com isso é possível gerar uma lista de eventos sobre aplicativos (i.e., aplicativos foram instalados ou desinstalados) para cada cliente e saber qual a última data que o cliente abriu o aplicativo do banco digital. Além disso, filtramos usuários cujos dispositivos móveis não reportaram nenhuma instalação, desinstalação ou acesso ao aplicativo de banco.

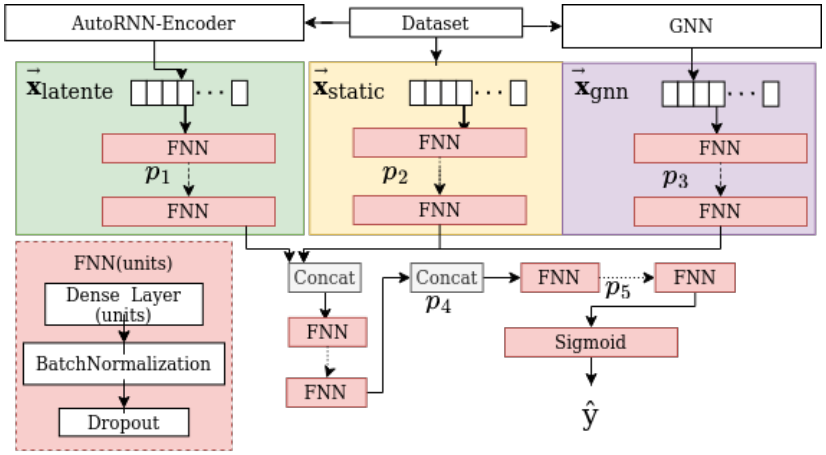


Figura 3: Arquitetura AutoRGNN.

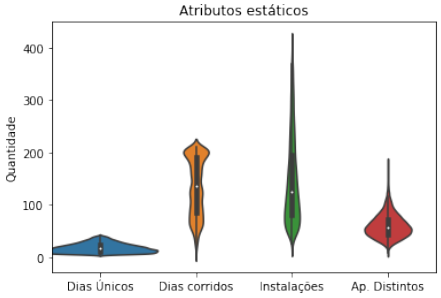


Figura 4: Distribuição de alguns atributos estáticos agregados por usuário. Da esquerda para a direta são representados os atributos: número de dias de registro, quantidade de dias do primeiro registro até o último dia, número de instalações totais e número de aplicativos distintos.

Os dados contemplam 47.665 clientes, coletados do período de 01-01-2021 a 30-07-2021. Além disso, foi possível coletar o último dia que o usuário acessou o aplicativo do banco digital em um período posterior, no dia 31-01-2022. Com isso, foram considerados *churners* clientes que ficaram 6 meses sem abrir o aplicativo (de 30-07-2021 até 31-01-2022) pois é desejável ter alta confiança na decisão de quais clientes serão classificados como *churners*. Esse intervalo de tempo maior foi utilizado, pois é possível que o banco digital estudado não seja o preferido do cliente. E por isso seja utilizado somente como conta de investimentos onde é feito um acesso de forma esporádica. Com isso, foi testada a hipótese de que o comportamento atual de *churn* utilizando a sequência de instalação e desinstalação de aplicativos pode anunciar uma tendência de *churn* no médio prazo (i.e., 6 meses). Isso pode ser interessante em corporações, pois pode ser necessário um tempo com engajamento de uma empresa para evitar o *churn* no futuro. Com esta definição de *churn*, foram registrados 11.504 *churners*de um total de 47.665 clientes.

Na figura 4 é possível ver as distribuições de alguns atributos do conjunto de dados. Além disso, ressaltamos alta esparsidade da representação de X_{rnn} definida na seção 3.1 com 99.86% dos dados nulos.

Para evitar sobre-ajustamento e acelerar o tempo de treinamento dos algoritmos, foram mantidos apenas os 5% aplicativos com maior número de instalações e desinstalações. Esse filtro manteve os 1.058 aplicativos mais frequentes do conjunto de dados. Além isso, foram removidos os clientes que não tenham pelo menos dois meses de dados gravados, pois a quantidade de eventos deles poderia ser insuficiente para o treinamento e/ou definição de *churn*.

Devido à baixa proporção de *churners*, foi necessário usar técnicas que permitem o aprendizado em conjuntos de dados desbalanceados. Foram encontrados os melhores resultados utilizando *undersampling* aleatório ao invés de uso de pesos em instâncias menos comuns nos modelos e outras técnicas mais avançadas de *resampling* (i.e., SMOTE, AdaSyn). Foi utilizado o *undersampling* aleatório eliminando aleatoriamente usuários não-churners até que o balanceamento 50%-50% fosse alcançado.

Para reduzir o número de preenchimento necessário nos tensores de treinamento da RNN, foram filtrados clientes com pelo menos 5 a 40 dias únicos. Isso foi feito pois os clientes com poucos dias distintos podem fazer com que a RNN Autocodificadora não tenha dados suficientes, pois não se tem conhecimento da sequência de eventos intra-diários. Além disso, clientes com muitos dias distintos também podem aumentar o sobre-ajustamento da rede, pois o número de células recorrentes precisará corresponder ao número de eventos para cada cliente. Portanto, foram filtrados clientes em que o número de eventos esteja no percentil 10% (55 eventos) e 95% (700 eventos).

4.2 Floresta Aleatória (Baseline)

Nesta seção é definida uma *baseline* utilizando algoritmos tradicionais de aprendizado de máquina para a previsão de *churn*. Para isso, foi utilizada a ferramenta de Aprendizado de Máquina Automatizado (AutoML) TPOT [12]. Essa ferramenta otimiza grande parte do processo necessário para utilização de modelos de aprendizado de máquina. O TPOT utiliza algoritmos genéticos para escolher qual o pré-processamento, construção e seleção de atributos, algoritmos e

hiper-parâmetros que produzem o melhor modelo conforme métricas selecionadas pelo usuário. O uso dessa ferramenta permite a avaliação com validação cruzada de diversos modelos tradicionais de aprendizado de máquina. Isso aumenta a chance de se ter um modelo eficiente, e com isso uma comparação mais justa com a proposta deste trabalho.

Já que os modelos tradicionais de aprendizado de máquina disponíveis pelo TPOT não são algoritmos adaptados para dados sequenciais como a RNN, no treinamento da *baseline* foi utilizada somente a lista de aplicativos com codificação binária (i.e., *One Hot Encoding*). Esta lista de aplicativos é a disponível no último dia de registro no conjunto de dados para cada cliente. Além disso, também são fornecidas as mesmas informações estáticas calculadas a partir da sequência de instalação e desinstalação, definidas na seção 3.3. Como as categorias dos aplicativos são utilizadas na solução proposta neste trabalho, a *baseline* também tem acesso à quantidade de aplicativos instalados distribuído por categoria. Já como objetivo da classificação, como rótulo de cada exemplo é utilizado o indicador de *churn* definido na seção 4.1.

O algoritmo selecionado pelo TPOT foi a Floresta Aleatória (*Random Forest*), o que faz sentido, pois, trata-se de um problema esparso com múltiplas colunas categóricas e com outras colunas com codificação numérica. Nesse caso a Floresta Aleatória é um algoritmo com potencial de extrair informações lineares e não-lineares com um baixo sobre-ajustamento. O uso do TPOT nos permite ter uma solução base competitiva com as melhores configurações encontradas após várias iterações do algoritmo genético. Além disso, é importante ressaltar que, além da Floresta Aleatória, o TPOT não selecionou nenhum outro método como, por exemplo, seleção de atributos, redução de dimensionalidade, *stacking* ou *bagging*.

Quanto aos hiperparâmetros, a Floresta Aleatória utilizou os seguintes: uso de *bootstrap*, critério de divisão Gini, percentual dos melhores atributos como sendo 0,2. Uma descrição desses atributos pode ser encontrada na documentação da biblioteca utilizada[3]. Esses parâmetros foram encontrados após a busca pelo TPOT utilizando 20 gerações com população igual a 100 e 5 divisões na validação cruzada utilizando a média do *F1-score*.

4.3 Configuração

Foram utilizadas as bibliotecas Keras[4], Tensorflow[5] e Spektral[6] para as implementações (i.e., treinamento e derivações automáticas) da solução proposta AutoRGNN. Em relação aos hiperparâmetros, foram utilizados: a taxa de aprendizado com valor de 0,01, algoritmo de otimização Adamax [11], tamanho do *batch* como 32, foi inicializada com pesos aleatórios utilizando método de Glorot, *EarlyStopping* com 10 de paciência, redução dinâmica de taxa de aprendizado com paciência de 8. Já para as ativações de todas camadas foi utilizada a ReLU (*Rectified Linear Unit*), com exceção da última, com ativação *sigmoid* devido à classificação binária.

Além disso, empiricamente é notável a necessidade de utilização de camadas bidirecionais ao invés de unidirecionais para uma boa acurácia do modelo. Também foi percebido que o uso de células GRU resultaram em acurácias parecidas com células LSTM, mas com a vantagem de convergirem mais rapidamente. Além disso, quantidade de unidades (i.e., neurônios) em cada camada dos modelos foi otimizada manualmente utilizando um conjunto de validação contendo 20% dos dados. Já para a quantidade de unidades (i.e., células) em cada camada dos modelos e profundidade (i.e., quantidade de camadas empilhadas) nos modelos Auto RNN, GNN e AutoRGNN foram determinados utilizando otimização bayesiana com auxílio da ferramenta *Keras Tuner* [13], utilizando hiperparâmetros $\alpha = 0,0001$ e $\beta = 2,6$. Com isso, foram necessárias 300 avaliações de conjuntos de hiperparâmetros selecionados pela otimização bayesiana para obter os resultados deste trabalho. Além disso, o método de redução de dimensionalidade descrito na seção 3.3 resultou na escolha de 50 aplicativos. Também foi percebido que, o uso de dados estáticos além dos aplicativos instalados, melhorou consideravelmente a acurácia dos modelos.

Devido à complexidade computacional, a escolha dos melhores hiperparâmetros foi feita utilizando o método de *holdout* simples separando 20% dos usuários para avaliar o melhor conjunto de hiperparâmetros . Foram feitos os testes dos modelos utilizando validação cruzada com 5 divisões e 5 repetições. Em cada iteração, foram separados 10% dos usuários de treinamento para a validação do modelo. A repetição da validação cruzada é necessária devido a principalmente a possibilidade dos algoritmos otimizados pelo *Adamax* atingirem um ótimo local. Por isso, em cada *fold* da validação são feitas 5 avaliações com inicializações diferentes dos modelos, sendo utilizado para teste o modelo que obteve o maior valor no conjunto de validação.

4.4 Resultados

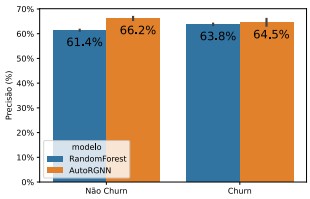
A Figura 5a ilustra a maior capacidade da AutoRGNN para prever tanto *churners* quanto não *churners*, uma vez que a precisão é maior em ambas as situações. No caso da precisão é percebido um aumento de 7% na média da precisão ao prever não *churners*; já para os *churners* o aumento na média é mais discreto (1%) e menos significativo devido ao maior desvio padrão da AutoRGNN. Já para a revocação (veja Figura 5b), existe uma tendência do modelo proposto a detectar melhor os usuários *churners*, com o aumento de 19% na revocação. Porém, é percebida uma menor certeza da predição de não *churners* com a diminuição da revocação em 7%. É importante destacar que, na prática, para a previsão de *churn* é melhor um falso positivo do que um falso negativo devido a possível perda de um cliente ao realizar *churn*.

No geral, a figura 5c mostra que é possível melhorar a previsão de *churn* de serviços móveis mesmo utilizando dados altamente esparsos e não invasivos. Destacamos que, em geral, a previsão de *churn* sem uso de informações específicas do negócio é um problema difícil, pois, em cada contexto, existe uma forma de definir *churn* e comportamentos específicos na plataforma digital.

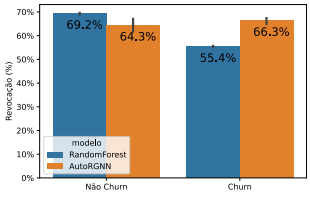
5 CONSIDERAÇÕES FINAIS

Neste trabalho observa-se que modelos de aprendizado profundo podem apresentar melhor resultado que modelos tradicionais para previsão de *churn* em ambientes que não permitem uso de dados sensíveis dos clientes. Além disso, é importante observar que a

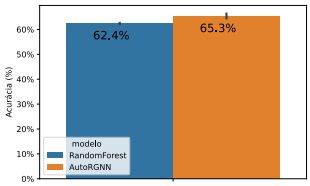
[3] https://scikit-learn.org/0.24/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[4] https://keras.io/
[5] https://www.tensorflow.org
[6] https://graphneural.network/



(a) Média e desvio padrão da precisão entre usuários *churners* e não *churners* para ambos modelos.



(b) Média e desvio padrão da revocação entre usuários *churners* e não *churners* para ambos modelos.



(c) Média e desvio padrão da acurácia para ambos modelos.

Figura 5: Comparação de métricas para os modelos de predição de *churn*.

solução proposta, chamada AutoRGNN, pode ser usada para codificação de ainda mais informações dos clientes na forma de grafos e sequências, o que pode melhorar ainda mais o desempenho dos modelos.

Como trabalhos futuros, pretende-se validar o modelo proposto utilizando outro estudo de caso não relacionado a bancos digitais. Com isso, será possível verificar a generalização da proposta. Além disso, para mais uma forma de comparação, pretende-se implementar alguma solução de aprendizado de máquina tradicional que possa aproveitar das informações sequenciais, como as Cadeias de Markov, em combinação com a Floresta Aleatória.

REFERÊNCIAS

[1] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *CoRR* abs/1412.3555 (2014). arXiv:1412.3555 http://arxiv.org/abs/1412.3555

[2] Abigayle Erickson. 2018. Comparative Analysis of the EU's GDPR and Brazil's LGPD: Enforcement Challenges with the LGPD. *Brook. J. Int'l L.* 44 (2018), 859.

[3] David L García, Àngela Nebot, and Alfredo Vellido. 2017. Intelligent data analysis approaches to churn as a business problem: a survey. *Knowledge and Information Systems* 51, 3 (2017), 719–774.

[4] Lara Rocha Garcia, Edson Aguilera-Fernandes, Rafael Augusto Moreno Gonçalves, and Marcos Ribeiro Pereira-Barretto. 2020. *Lei Geral de Proteção de Dados (LGPD): guia de implantação.* Editora Blucher.

[5] Sunil Gupta, Donald R Lehmann, and Jennifer Ames Stuart. 2004. Valuing customers. *Journal of marketing research* 41, 1 (2004), 7–18.

[6] Seyed Jamal Haddadi, Mohammad Ostad Mohammadi, Mojtaba Bahrami, Elham Khoeini, Mehdi Beygi, and Mehrdad Haddad Khoshkar. 2022. Customer Churn Prediction in the Iranian Banking Sector. In *2022 International Conference on Applied Artificial Intelligence (ICAPAI)*. 1–6. https://doi.org/10.1109/ICAPAI55158.2022.9801574

[7] Niels Holtrop, Jaap E. Wieringa, Maarten J. Gijsenberg, and Peter C. Verhoef. 2017. No future without the past? Predicting churn in the face of customer privacy. *International Journal of Research in Marketing* 34, 1 (2017), 154–172. https://doi.org/10.1016/j.ijresmar.2016.06.001

[8] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. 2016. Densely Connected Convolutional Networks. *CoRR* abs/1608.06993 (2016). arXiv:1608.06993 http://arxiv.org/abs/1608.06993

[9] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 37)*, Francis Bach and David Blei (Eds.). PMLR, Lille, France, 448–456. https://proceedings.mlr.press/v37/ioffe15.html

[10] Hemlata Jain, Ajay Khunteta, and Sumit Srivastava. 2020. Churn prediction in telecommunication using logistic regression and logit boost. *Procedia Computer Science* 167 (2020), 101–112.

[11] Diederik P Kingma and Jimmy Ba. 2014. Adam: a method for stochastic optimization (2014). *arXiv preprint arXiv:1412.6980* 22 (2014).

[12] Randal S. Olson and Jason H. Moore. 2016. TPOT: A Tree-based Pipeline Optimization Tool for Automating Machine Learning. In *Proceedings of the Workshop on Automatic Machine Learning (Proceedings of Machine Learning Research, Vol. 64)*, Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren (Eds.). PMLR, New York, New York, USA, 66–74. https://proceedings.mlr.press/v64/olson_tpot_2016.html

[13] Tom O'Malley, Elie Bursztein, James Long, François Chollet, Haifeng Jin, Luca Invernizzi, et al. 2019. KerasTuner. https://github.com/keras-team/keras-tuner.

[14] Manas Rahman and V Kumar. 2020. Machine Learning Based Customer Churn Prediction In Banking. In *2020 4th International Conference on Electronics, Communication and Aerospace Technology (ICECA)*. 1196–1201. https://doi.org/10.1109/ICECA49313.2020.9297529

[15] R Rajamohamed and J Manokaran. 2018. Improved credit card churn prediction based on rough clustering and supervised learning techniques. *Cluster Computing* 21, 1 (2018), 65–77.

[16] Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing* 45, 11 (1997), 2673–2681.

[17] Fei Tan, Zhi Wei, Jun He, Xiang Wu, Bo Peng, Haoran Liu, and Zhenyu Yan. 2018. A Blended Deep Learning Approach for Predicting User Intended Actions. In *2018 IEEE International Conference on Data Mining (ICDM)*. 487–496. https://doi.org/10.1109/ICDM.2018.00064

[18] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2017. Graph Attention Networks. https://doi.org/10.48550/ARXIV.1710.10903

[19] Théo Verhelst. 2018. Churn Prediction and Causal Analysis on Telecom Customer Data. (2018).

[20] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. 2013. Regularization of Neural Networks using DropConnect. In *Proceedings of the 30th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 28)*, Sanjoy Dasgupta and David McAllester (Eds.). PMLR, Atlanta, Georgia, USA, 1058–1066. https://proceedings.mlr.press/v28/wan13.html

[21] Shuting Xu, Shuhua Lai, and Manying Qiu. 2009. Privacy Preserving Churn Prediction. In *Proceedings of the 2009 ACM Symposium on Applied Computing* (Honolulu, Hawaii) *(SAC '09)*. Association for Computing Machinery, New York, NY, USA, 1610–1614. https://doi.org/10.1145/1529282.1529643

[22] Carl Yang, Xiaolin Shi, Luo Jie, and Jiawei Han. 2018. I Know You'll Be Back: Interpretable New User Clustering and Churn Prediction on a Mobile Social Application. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (London, United Kingdom) *(KDD '18)*. Association for Computing Machinery, New York, NY, USA, 914–922. https://doi.org/10.1145/3219819.3219821

[23] Jie Zhou, Jian-feng Yan, Lu Yang, Meng Wang, and Peng Xia. 2019. Customer churn prediction model based on lstm and cnn in music streaming. *DEStech Transactions on Engineering and Technology Research* 5 (2019).